Fuzzing 101

Whoami

- Bloomsburg University - Computer Science
- But I have always had an interest in cyber security and hacking
  - Finding ways to make computers do things that they were not meant to do
  - So that we can get the issues fixed before the attackers discover them and try to abuse them
- Dakota State University - Masters in Information Assurance
- GWAPT, GPEN, GXPN certifications
- My day job is penetration testing for an IT company
- In my free time after work and on weekends, I do security and exploit research

What is fuzzing?

- Sending input to an application, that the application is not expecting, to try to make it crash
  - Long strings of characters that might overflow a memory buffer
  - Weird characters that the application isn’t expecting
  - Malformed input files

Why fuzz?

- I wanted to learn about fuzzing, to assist with my learning about exploit development
- Exploitable code is commonly found via fuzzing
  - Especially if you don't have access to the source code
- We will be focusing on binary fuzzing
  - Web app fuzzing is another topic (although very common)

Why fuzz?

- Some deep-in-code bugs are being found, that have been there for many years
  - I was curious about how they were being found
  - And why they weren't found sooner
- Basically, the only way to find these deep-in-code bugs is thru fuzzing
  - It is all but impossible to manually test all the input combinations

Why fuzz?

- Many security testers and researchers use fuzzing to find these deep-in-code vulnerabilities
  - Many software companies have security testers who fuzz their products before they are released to market
  - Google Project Zero is famous for fuzzing many companies' products, to catch remaining bugs
    - Other research organizations do this as well
    - Provides a 2nd set of eyes
  - Many open-source projects are also being fuzzed by security researchers, to find and fix bugs in them

Disclaimer

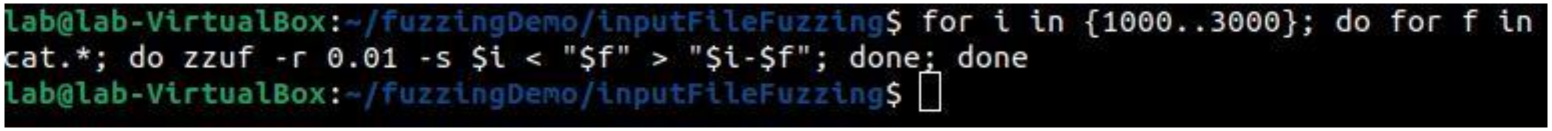
- Note that this is a very basic primer of fuzzing
    - You can go much deeper and further, expanding code coverage, etc
    - But my goal right now was just to learn the basics

Malformed input file fuzzing

- Using the Zzuf tool, make random small changes to a file. The resulting file can then be fed into an application, to try to make it crash.
- Image files
- Documents, Pdf files
- Etc

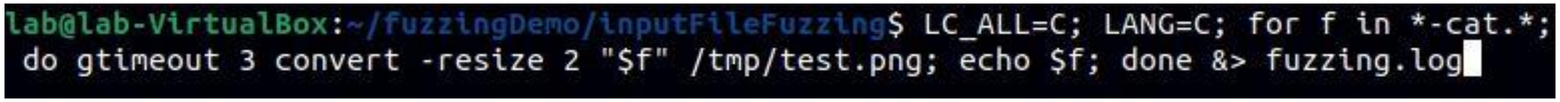
Malformed input file fuzzing

- 1) Modify some pictures a bunch of times
  - cat.* is cat pictures, not the linux cat command



```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ for i in {1000..3000}; do for f in
cat.*; do zzuf -r 0.01 -s $i < "$f" > "$i-$f"; done; done
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$
```

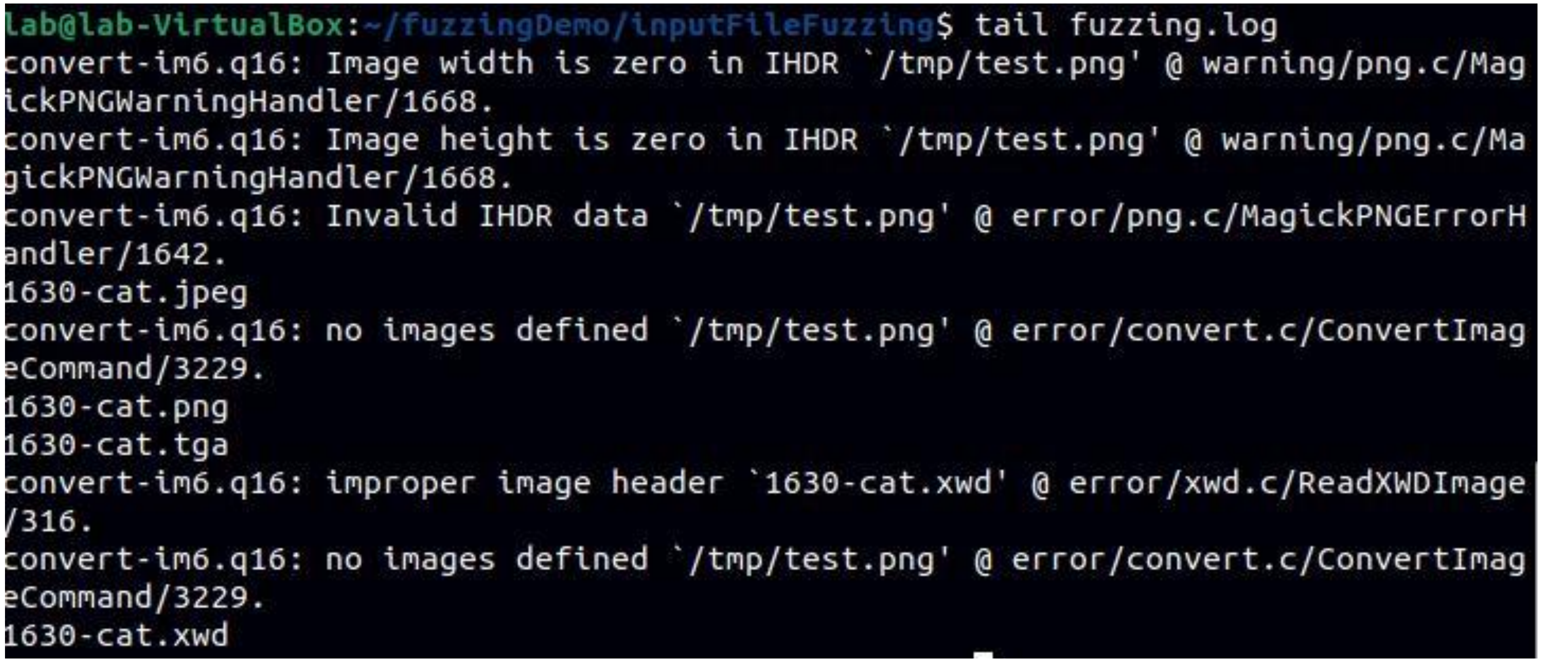
- 2) Feed the modified pictures into an image converter, save error messages to log file
  - Image Magick, any others



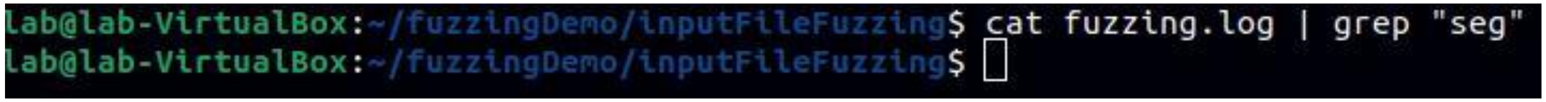
```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ LC_ALL=C; LANG=C; for f in *-cat.*;
 do gtimeout 3 convert -resize 2 "$f" /tmp/test.png; echo $f; done &> fuzzing.log
```

Malformed input file fuzzing

- 3) Look in the log file for segmentation faults



```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ tail fuzzing.log
convert-im6.q16: Image width is zero in IHDR `/tmp/test.png' @ warning/png.c/Mag
ickPNGWarningHandler/1668.
convert-im6.q16: Image height is zero in IHDR `/tmp/test.png' @ warning/png.c/Ma
gickPNGWarningHandler/1668.
convert-im6.q16: Invalid IHDR data `/tmp/test.png' @ error/png.c/MagickPNGErrorH
andler/1642.
1630-cat.jpeg
convert-im6.q16: no images defined `/tmp/test.png' @ error/convert.c/ConvertImag
eCommand/3229.
1630-cat.png
1630-cat.tga
convert-im6.q16: improper image header `1630-cat.xwd' @ error/xwd.c/ReadXWDImage
/316.
convert-im6.q16: no images defined `/tmp/test.png' @ error/convert.c/ConvertImag
eCommand/3229.
1630-cat.xwd
```



```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ cat fuzzing.log | grep "seg"
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ 
```

Malformed text entry fuzzing

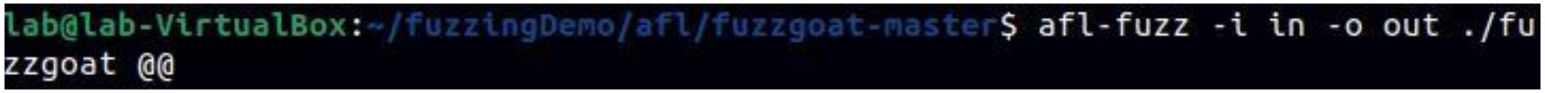
- Using AFL (American Fuzzy Lop)
  - Also the name of a rabbit breed, if you search for help for the tool
- It takes text input via input files, and then morphs it to try to break things
- It also tries to spider out through the code to hit all paths and get good coverage

Malformed text entry fuzzing

- Warnings:
- It requires at least 1 input that does not cause an error
- And if an input causes an error, it will not try to morph that input further
  - bad json file = won't try to further morph that json

Malformed text entry fuzzing

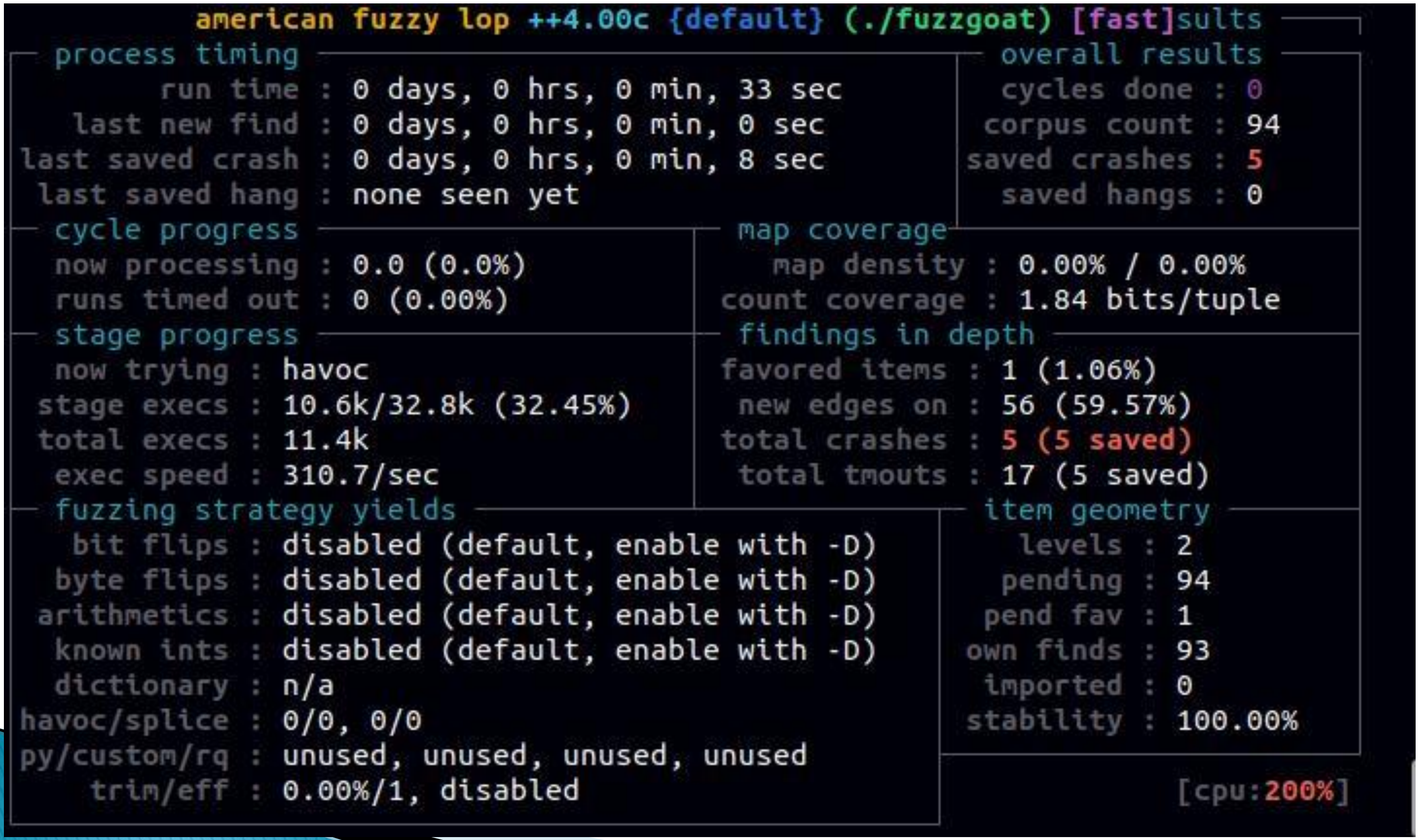
- Run AFL on the binary
  - Specifying the directory that contains the input files
  - And the directory that AFL should put the output files to



```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master$ afl-fuzz -i in -o out ./fu
zzgoat @@
```

Malformed text entry fuzzing

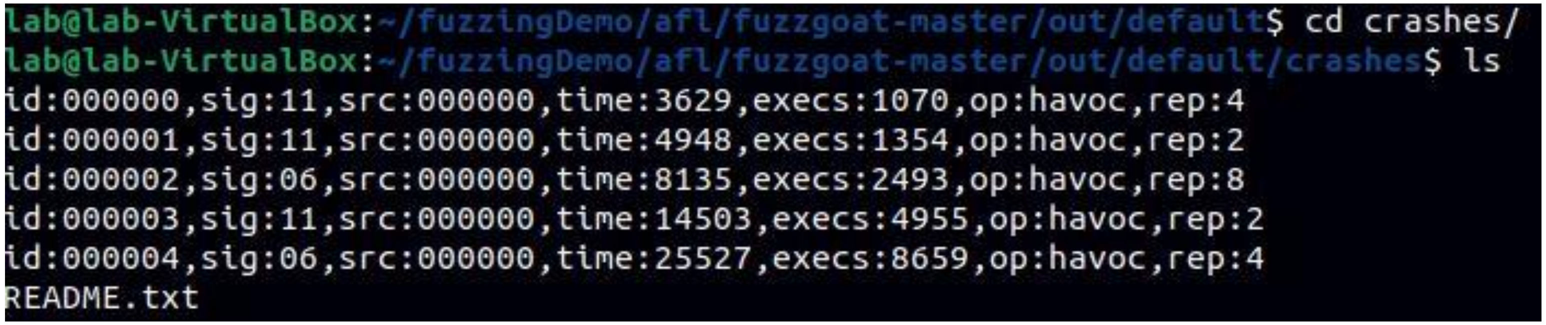
- AFL will show its progress
  - Note the total crashes counter in red on the right



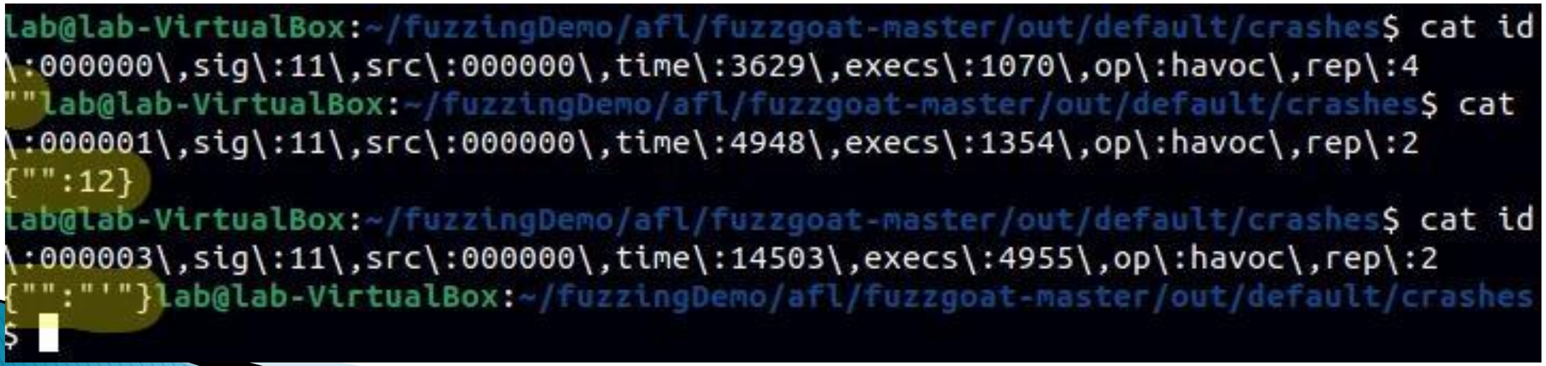
```
           american fuzzy lop ++4.00c {default} (./fuzzgoat) [fast]sults
┌─ process timing ─────────────────────────────────────┬─ overall results ──┐
│        run time : 0 days, 0 hrs, 0 min, 33 sec       │  cycles done : 0   │
│   last new find : 0 days, 0 hrs, 0 min, 0 sec        │ corpus count : 94  │
│last saved crash : 0 days, 0 hrs, 0 min, 8 sec        │saved crashes : 5   │
│ last saved hang : none seen yet                      │  saved hangs : 0   │
├─ cycle progress ───────────────────┬─ map coverage─┴──────────────────────┤
│  now processing : 0.0 (0.0%)       │    map density : 0.00% / 0.00%       │
│  runs timed out : 0 (0.00%)        │ count coverage : 1.84 bits/tuple     │
├─ stage progress ───────────────────┼─ findings in depth ──────────────────┤
│  now trying : havoc                │ favored items : 1 (1.06%)            │
│ stage execs : 10.6k/32.8k (32.45%) │  new edges on : 56 (59.57%)          │
│ total execs : 11.4k                │ total crashes : 5 (5 saved)          │
│  exec speed : 310.7/sec            │  total tmouts : 17 (5 saved)         │
├─ fuzzing strategy yields ──────────┴─────────────┬─ item geometry ────────┤
│   bit flips : disabled (default, enable with -D) │    levels : 2          │
│  byte flips : disabled (default, enable with -D) │   pending : 94         │
│ arithmetics : disabled (default, enable with -D) │  pend fav : 1          │
│  known ints : disabled (default, enable with -D) │ own finds : 93         │
│  dictionary : n/a                                │  imported : 0          │
│havoc/splice : 0/0, 0/0                           │ stability : 100.00%    │
│py/custom/rq : unused, unused, unused, unused     ├────────────────────────┘
│    trim/eff : 0.00%/1, disabled                  │          [cpu:200%]
└──────────────────────────────────────────────────┘
```

Malformed text entry fuzzing

- AFL will put the crashes in files in the out/default/crashes directory
  - Each file will contain an input that caused a crash



```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default$ cd crashes/
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ ls
id:000000,sig:11,src:000000,time:3629,execs:1070,op:havoc,rep:4
id:000001,sig:11,src:000000,time:4948,execs:1354,op:havoc,rep:2
id:000002,sig:06,src:000000,time:8135,execs:2493,op:havoc,rep:8
id:000003,sig:11,src:000000,time:14503,execs:4955,op:havoc,rep:2
id:000004,sig:06,src:000000,time:25527,execs:8659,op:havoc,rep:4
README.txt
```



```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat id
\:000000\,sig\:11\,src\:000000\,time\:3629\,execs\:1070\,op\:havoc\,rep\:4
""lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat
\:000001\,sig\:11\,src\:000000\,time\:4948\,execs\:1354\,op\:havoc\,rep\:2
{"":12}
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat id
\:000003\,sig\:11\,src\:000000\,time\:14503\,execs\:4955\,op\:havoc\,rep\:2
{"":"'"}lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes
$ 
```

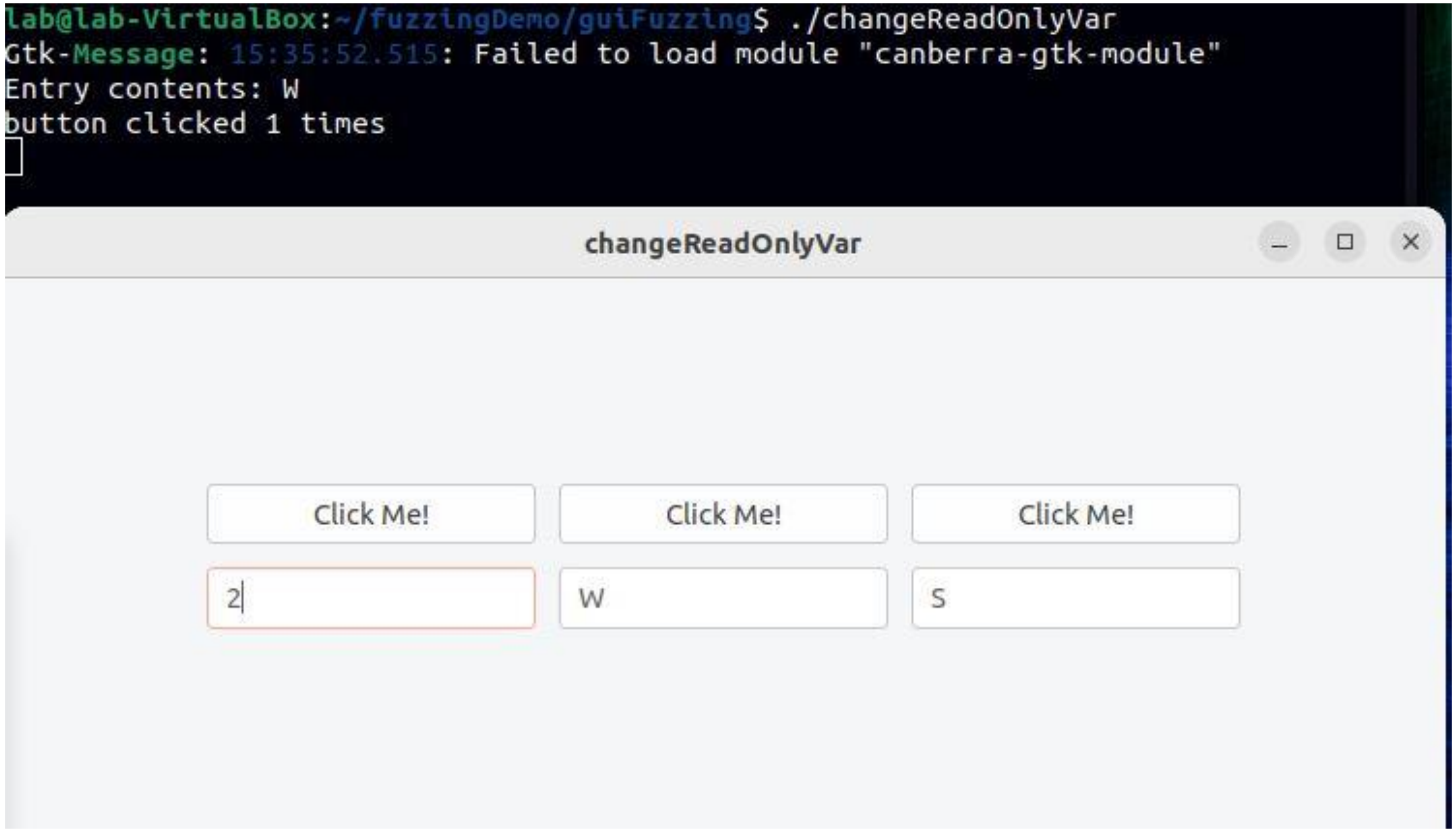
Gui fuzzing

- Not straight-forward like sending a command line argument or input file to an application
- Instead have to simulate mouse clicks and keyboard strokes
  - To random places
  - Or to specific spots
- No standard fuzzing tools available, since it needs to be so customized

Gui fuzzing – Demo

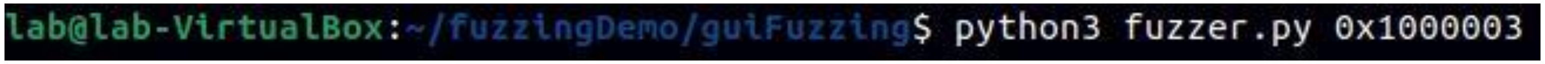
- I made a simple python-based fuzzer
  - That clicks at random places within an application’s window
  - And tabs through the application’s gui elements, and sends random characters
- Poc vulnerable c program
  - That does a segfault when the left button is clicked 3x
  - Or when “3” is entered into the middle textbox

Gui fuzzing – Demo

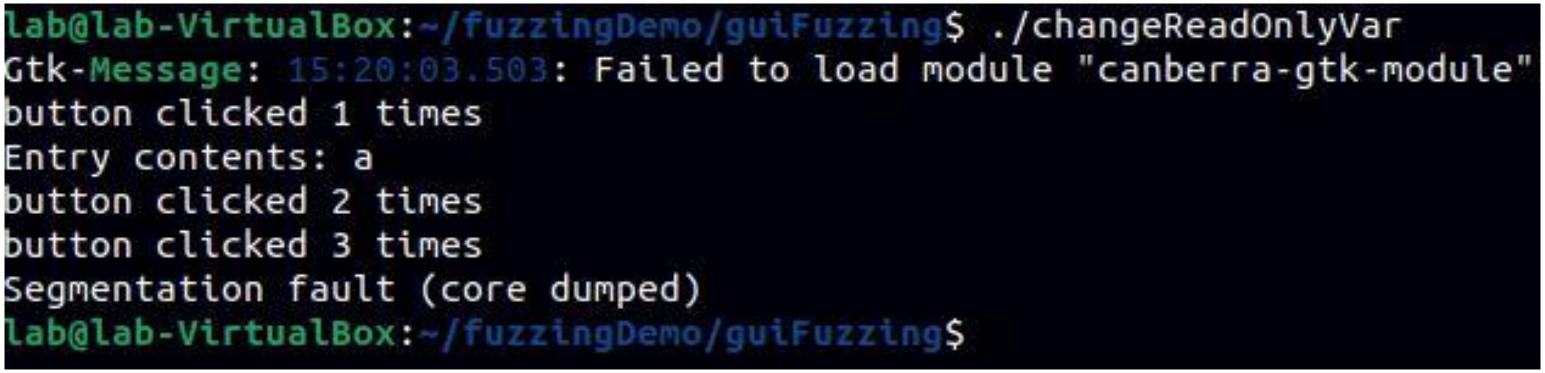


```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ ./changeReadOnlyVar
Gtk-Message: 15:35:52.515: Failed to load module "canberra-gtk-module"
Entry contents: W
button clicked 1 times
```

Gui fuzzing – Demo

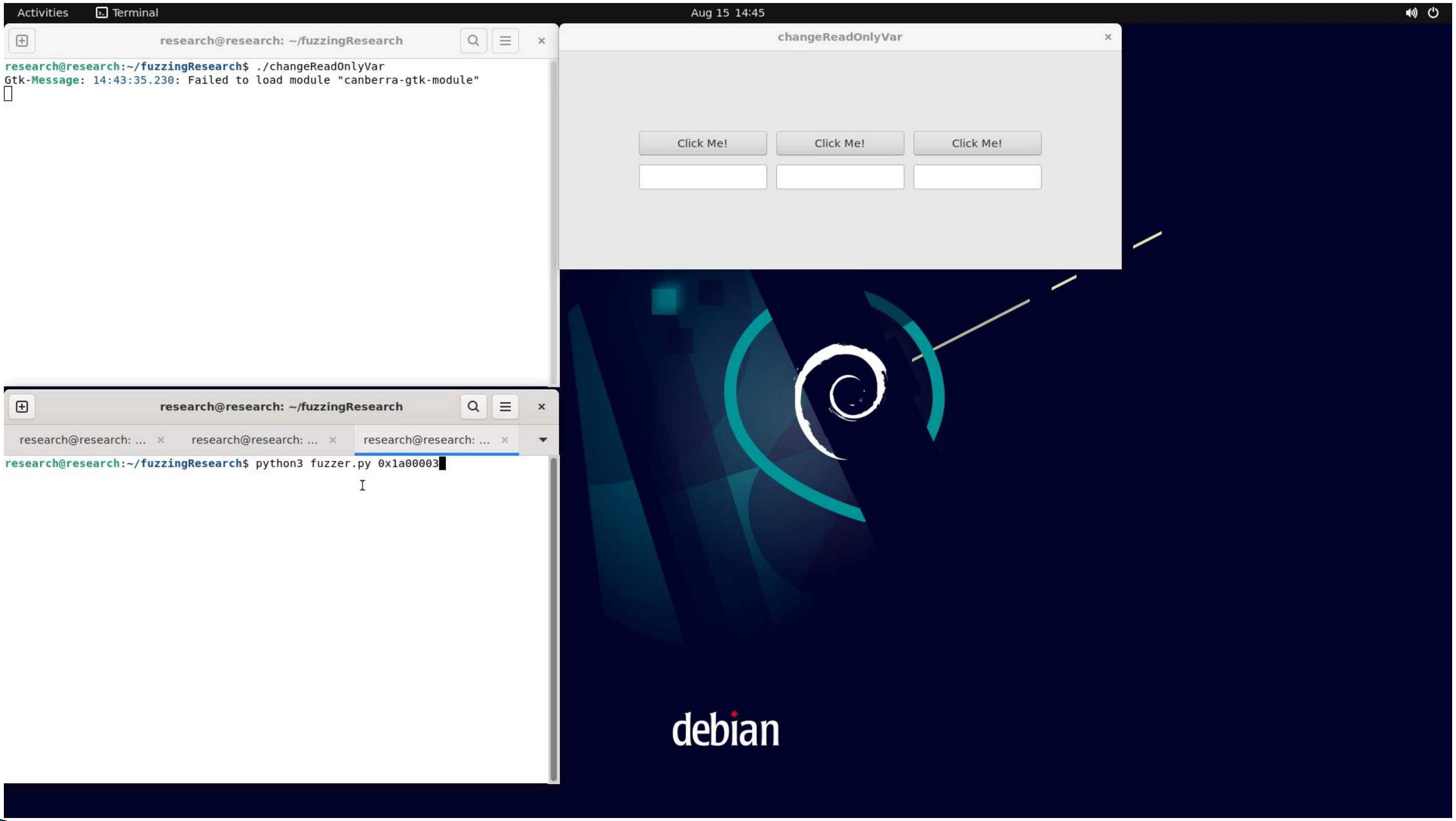


```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ python3 fuzzer.py 0x1000003
```



```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ ./changeReadOnlyVar
Gtk-Message: 15:20:03.503: Failed to load module "canberra-gtk-module"
button clicked 1 times
Entry contents: a
button clicked 2 times
button clicked 3 times
Segmentation fault (core dumped)
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$
```

Gui fuzzing – Demo



```
research@research:~/fuzzingResearch$ ./changeReadOnlyVar
Gtk-Message: 14:43:35.230: Failed to load module "canberra-gtk-module"
```

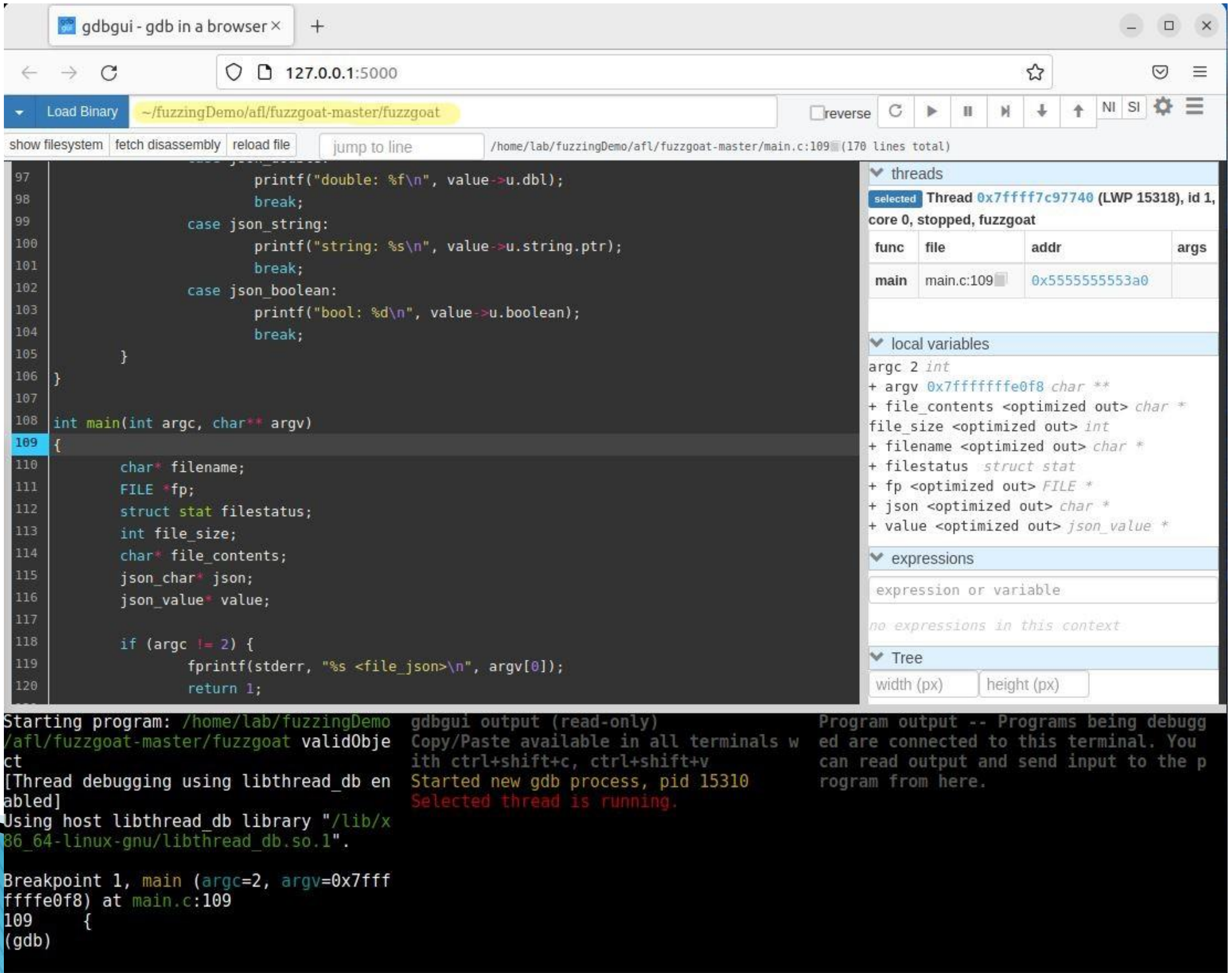
```
research@research:~/fuzzingResearch$ python3 fuzzer.py 0x1a00003
```

Debugging

- Not every crash / segfault is exploitable
  - There could be runtime or memory protections that prevent you from taking further control
- The next step is to run the application through a debugger
  - To see what code section the application crashes in
  - To then see how you may be able to further exploit the code
  - Use the malformed input that caused the crash

Debugging

- Load the target binary


gdbgui - gdb in a browser
127.0.0.1:5000
Load Binary
~/fuzzingDemo/afl/fuzzgoat-master/fuzzgoat
reverse
NI
SI
show filesystem
fetch disassembly
reload file
jump to line
/home/lab/fuzzingDemo/afl/fuzzgoat-master/main.c:109 (170 lines total)
threads
expressions
expression or variable
no expressions in this context
Tree
width (px)
height (px)

```c
                printf("double: %f\n", value->u.dbl);
                break;
            case json_string:
                printf("string: %s\n", value->u.string.ptr);
                break;
            case json_boolean:
                printf("bool: %d\n", value->u.boolean);
                break;
        }
}

int main(int argc, char** argv)
{
        char* filename;
        FILE *fp;
        struct stat filestatus;
        int file_size;
        char* file_contents;
        json_char* json;
        json_value* value;

        if (argc != 2) {
                fprintf(stderr, "%s <file_json>\n", argv[0]);
                return 1;
```

selected Thread 0x7ffff7c97740 (LWP 15318), id 1, core 0, stopped, fuzzgoat

func	file	addr	args
main	main.c:109	0x5555555553a0	

local variables

```
argc 2 int
+ argv 0x7fffffffe0f8 char **
+ file_contents <optimized out> char *
file_size <optimized out> int
+ filename <optimized out> char *
+ filestatus  struct stat
+ fp <optimized out> FILE *
+ json <optimized out> char *
+ value <optimized out> json_value *
```

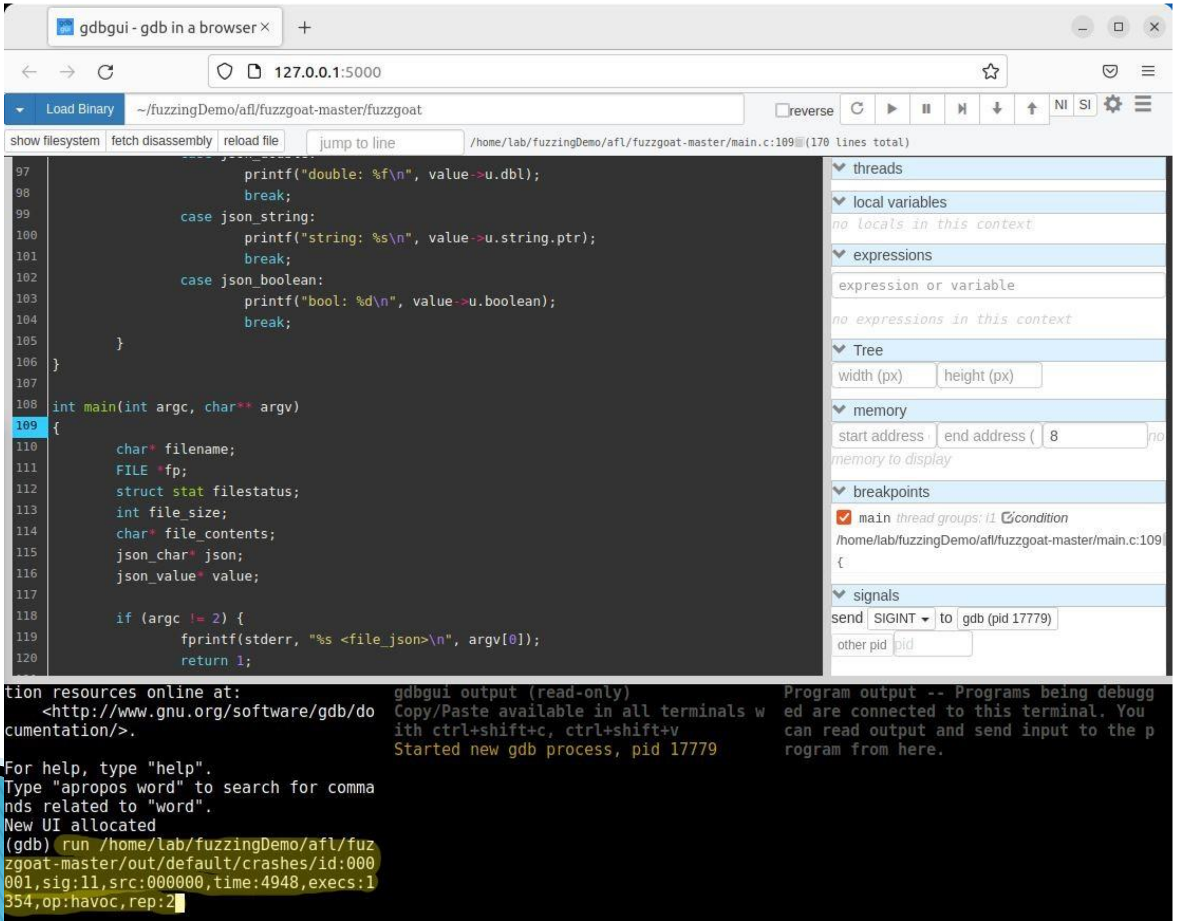
```
Starting program: /home/lab/fuzzingDemo
/afl/fuzzgoat-master/fuzzgoat validObje
ct
[Thread debugging using libthread_db en
abled]
Using host libthread_db library "/lib/x
86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=2, argv=0x7fff
ffffe0f8) at main.c:109
109     {
(gdb)
```

```
gdbgui output (read-only)
Copy/Paste available in all terminals w
ith ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 15310
Selected thread is running.
```

```
Program output -- Programs being debugg
ed are connected to this terminal. You
can read output and send input to the p
rogram from here.
```

Debugging

- Run the target binary, with the arguments that cause the crash



~/fuzzingDemo/afl/fuzzgoat-master/fuzzgoat

/home/lab/fuzzingDemo/afl/fuzzgoat-master/main.c:109 (170 lines total)

```
            printf("double: %f\n", value->u.dbl);
            break;
        case json_string:
            printf("string: %s\n", value->u.string.ptr);
            break;
        case json_boolean:
            printf("bool: %d\n", value->u.boolean);
            break;
    }
}

int main(int argc, char** argv)
{
    char* filename;
    FILE *fp;
    struct stat filestatus;
    int file_size;
    char* file_contents;
    json_char* json;
    json_value* value;

    if (argc != 2) {
        fprintf(stderr, "%s <file_json>\n", argv[0]);
        return 1;
```

no locals in this context

expression or variable

no expressions in this context

no memory to display

main thread groups: i1
/home/lab/fuzzingDemo/afl/fuzzgoat-master/main.c:109
{

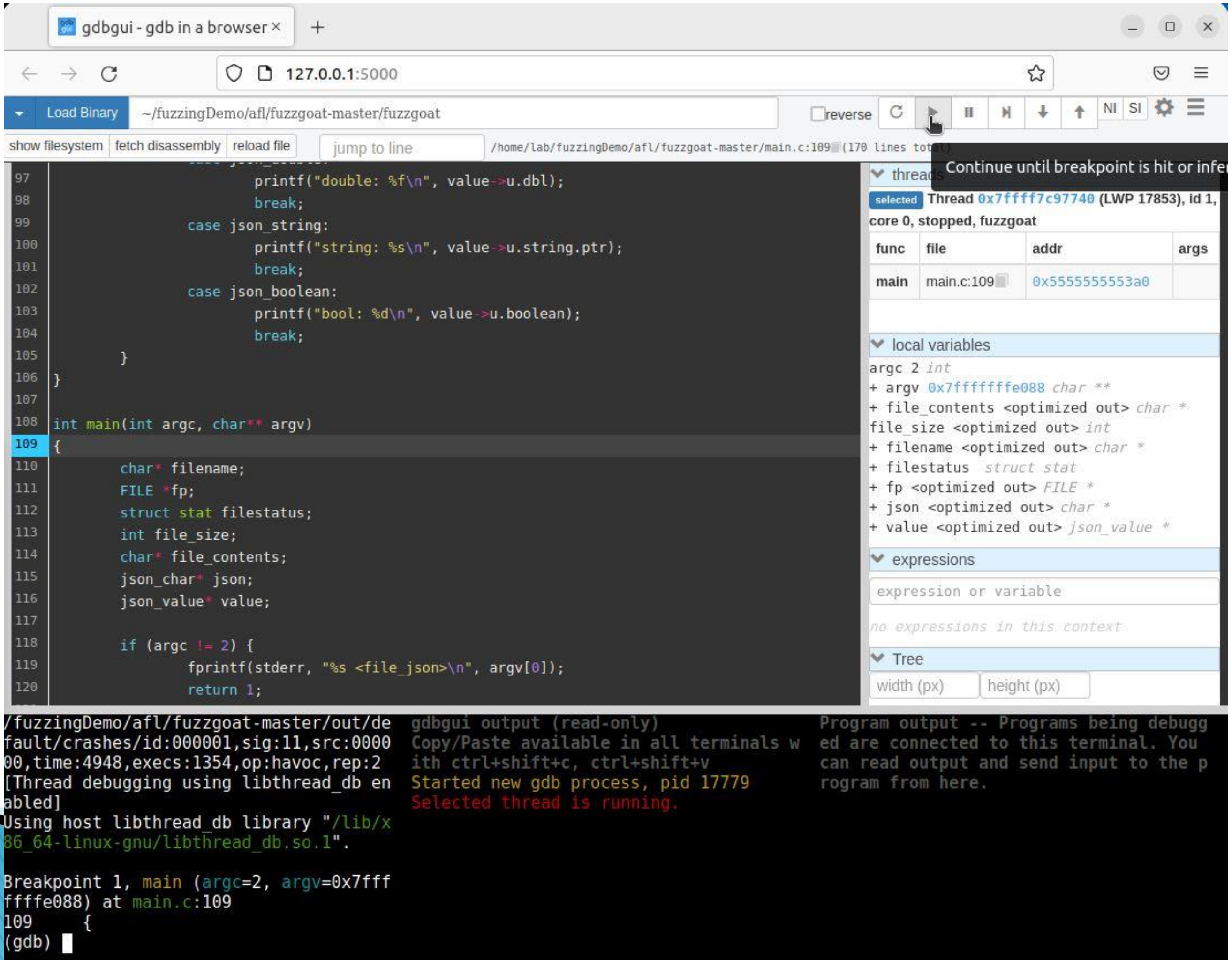
```
tion resources online at:
    <http://www.gnu.org/software/gdb/do
cumentation/>.

For help, type "help".
Type "apropos word" to search for comma
nds related to "word".
New UI allocated
(gdb) run /home/lab/fuzzingDemo/afl/fuz
zgoat-master/out/default/crashes/id:000
001,sig:11,src:000000,time:4948,execs:1
354,op:havoc,rep:2
```

```
gdbgui output (read-only)
Copy/Paste available in all terminals w
ith ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 17779
```

```
Program output -- Programs being debugg
ed are connected to this terminal. You
can read output and send input to the p
rogram from here.
```

Debugging

- Click the play button, and let it run


gdbgui - gdb in a browser
127.0.0.1:5000
Load Binary
~/fuzzingDemo/afl/fuzzgoat-master/fuzzgoat
reverse
NI
SI
show filesystem
fetch disassembly
reload file
jump to line
/home/lab/fuzzingDemo/afl/fuzzgoat-master/main.c:109 (170 lines total)
Continue until breakpoint is hit or infe

```
97                  printf("double: %f\n", value->u.dbl);
98                  break;
99              case json_string:
100                 printf("string: %s\n", value->u.string.ptr);
101                 break;
102             case json_boolean:
103                 printf("bool: %d\n", value->u.boolean);
104                 break;
105         }
106 }
107
108 int main(int argc, char** argv)
109 {
110         char* filename;
111         FILE *fp;
112         struct stat filestatus;
113         int file_size;
114         char* file_contents;
115         json_char* json;
116         json_value* value;
117
118         if (argc != 2) {
119                 fprintf(stderr, "%s <file_json>\n", argv[0]);
120                 return 1;
```

threads

selected Thread 0x7ffff7c97740 (LWP 17853), id 1, core 0, stopped, fuzzgoat

func	file	addr	args
main	main.c:109	0x5555555553a0	

local variables

```
argc 2 int
+ argv 0x7fffffffe088 char **
+ file_contents <optimized out> char *
file_size <optimized out> int
+ filename <optimized out> char *
+ filestatus  struct stat
+ fp <optimized out> FILE *
+ json <optimized out> char *
+ value <optimized out> json_value *
```

expressions

expression or variable

no expressions in this context

Tree

width (px) height (px)

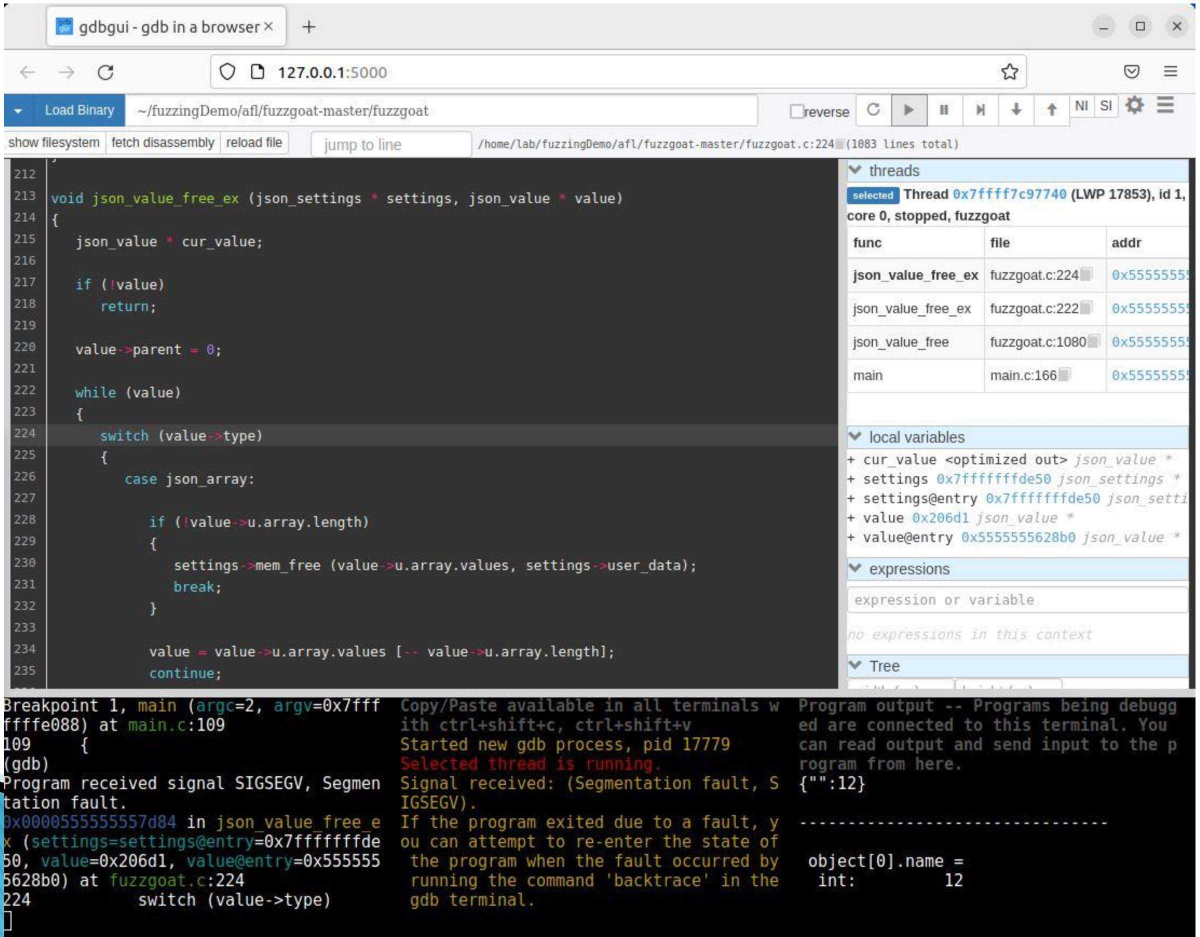
```
/fuzzingDemo/afl/fuzzgoat-master/out/de
fault/crashes/id:000001,sig:11,src:0000
00,time:4948,execs:1354,op:havoc,rep:2
[Thread debugging using libthread_db en
abled]
Using host libthread_db library "/lib/x
86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=2, argv=0x7fff
ffffe088) at main.c:109
109     {
(gdb) 
```

```
gdbgui output (read-only)
Copy/Paste available in all terminals w
ith ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 17779
Selected thread is running.
```

```
Program output -- Programs being debugg
ed are connected to this terminal. You
can read output and send input to the p
rogram from here.
```

Debugging

- When it crashes, it will stop at the line of code that caused the crash



```
/home/lab/fuzzingDemo/afl/fuzzgoat-master/fuzzgoat.c:224 (1083 lines total)
```

```
void json_value_free_ex (json_settings * settings, json_value * value)
{
   json_value * cur_value;

   if (!value)
      return;

   value->parent = 0;

   while (value)
   {
      switch (value->type)
      {
         case json_array:

            if (!value->u.array.length)
            {
               settings->mem_free (value->u.array.values, settings->user_data);
               break;
            }

            value = value->u.array.values [-- value->u.array.length];
            continue;
```

threads

selected Thread 0x7ffff7c97740 (LWP 17853), id 1, core 0, stopped, fuzzgoat

func	file	addr
json_value_free_ex	fuzzgoat.c:224	0x55555555
json_value_free_ex	fuzzgoat.c:222	0x55555555
json_value_free	fuzzgoat.c:1080	0x55555555
main	main.c:166	0x55555555

local variables

```
+ cur_value <optimized out> json_value *
+ settings 0x7fffffffde50 json_settings *
+ settings@entry 0x7fffffffde50 json_setti
+ value 0x206d1 json_value *
+ value@entry 0x5555555628b0 json_value *
```

expressions

expression or variable

no expressions in this context

Tree

```
Breakpoint 1, main (argc=2, argv=0x7fff
ffffe088) at main.c:109
109     {
(gdb)
Program received signal SIGSEGV, Segmen
tation fault.
0x0000555555557d84 in json_value_free_e
x (settings=settings@entry=0x7fffffffde
50, value=0x206d1, value@entry=0x555555
5628b0) at fuzzgoat.c:224
224             switch (value->type)
```

```
Copy/Paste available in all terminals w
ith ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 17779
Selected thread is running.
Signal received: (Segmentation fault, S
IGSEGV).
If the program exited due to a fault, y
ou can attempt to re-enter the state of
 the program when the fault occurred by
 running the command 'backtrace' in the
 gdb terminal.
```

```
Program output -- Programs being debugg
ed are connected to this terminal. You
can read output and send input to the p
rogram from here.
{"":12}

--------------------------------

 object[0].name =
  int:         12
```

Debugging

- You can then investigate what the code did to cause the crash
  - But that is another massive topic for another presentation

Note

- Sometimes research doesn’t turn out like you expect
  - I fuzzed several other image converters, expecting to get a bunch of crashes and segfaults
    - Since it is widely assumed that open source stuff if full of vulns
  - But they all errored out gracefully
- But don’t let a fear of potential failure keep you from trying something

Questions

?